# The Chroma Software System for Lattice QCD

B. Joo (UKQCD, Edinburgh), R. G. Edwards (LHPC, JLAB)

*School of Physics*
*Particle Physics Theory*

## Abstract

We describe aspects of the Chroma software system for lattice QCD calculations. Chroma is an open source C++ based software system developed using the software infrastructure of the U.S. SciDAC initiative. The system uses SciDAC packages such as QDP++ for data parallel operations, QMP for parallel communications and QIO for binary I/O. XML I/O is provided by wrappers around the open source libxml2 library. Chroma interfaces with output from the BAGEL assembler generator for optimised lattice fermion kernels on some architectures. The code can be run on workstations, clusters and the QCDOC supercomputer. The majority of the development effort was carried out at the JLAB with strong collaboration from UKQCD. As a case study we briefly describe its use in our algorithmic investigations of the Overlap Fermion Matrix.

## Introduction

We present the Chroma software system [1] for lattice QCD (LQCD) calculations. Chroma aims to provide a computational LQCD toolbox which is flexible, portable and efficient on a wide range of architectures from desktop workstations to large parallel computers including clusters, commercially available machines and also new architectures such as the QCDOC.

Primary development on Chroma started at the JLAB for the U.S. lattice community, in particular the LHPC collaboration [2], using software from the U.S. SciDAC initiative [3]. This effort has been joined by the UKQCD collaboration [4] who have been contributing to the SciDAC software effort on all levels. The result of this collaboration has not only produced useful software but also highlights the benefits of large-scale international collaboration and the open source development process.

In order to achieve the goals of flexibility, portability and efficiency Chroma relies on several layers of SciDAC and UKQCD software.

### The SciDAC Software Hierarchy

Towards the end of 2000, the U.S. Lattice community embarked on an ambitious project through the U.S. SciDAC initiative to standardise a set of software components in order to allow the effective exploitation of computing resources for lattice QCD. Three main levels of software infrastructure were defined

**QCD Message Passing (QMP):** This layer provides a message passing API that contains the communications primitives needed to perform lattice QCD calculations. It has been designed to take advantage of the specialised communication hardware of emerging architectures, such as the Serial Communications Unit (SCU) of the QCDOC and the specialised capabilities of Myrinet Network devices.

**Level 1 – QCD Linear Algebra (QLA):** The QLA layer defines operations to be performed at each site of the lattice. These primitives include $SU(3)$ matrix multiplications and multiplication of colour vectors by $SU(3)$ matrices.

**Level 2 – QCD Data Parallel (QDP):** This layer provides lattice–wide operations such as basic linear algebra for lattice–wide fields.

**Level 3 – Special optimised software:** This layer will provide portable interfaces to highly optimised and machine–dependent pieces of code such as assembly–coded Dslash operators or specialised solvers.

A recent addition to this hierarchy is the **QCD I/O (QIO)** sublayer which provides a record oriented I/O API. The records are packaged with the **LIME** framework which is a descendant of the **Direct Internet Message Encapsulation (DIME)** proposal devised originally for web service attachments.

We illustrate the hierarchy in Fig. 1. Software for QMP, QLA, QDP and QIO are available from the SciDAC web site [3]. Chroma currently relies on the QMP layer, and a C++ implementation of the QDP layer called QDP++.

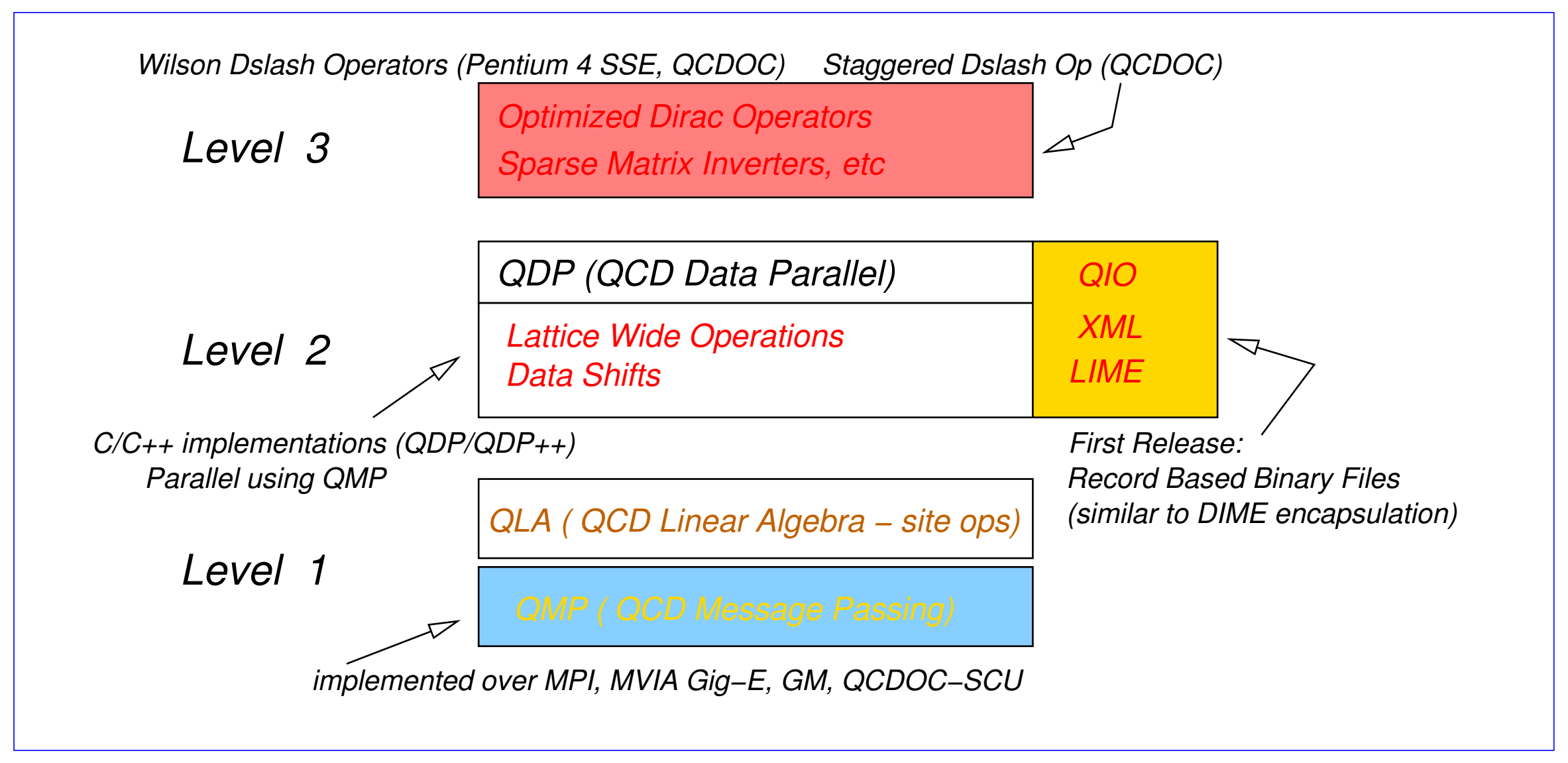
Wilson Dslash Operators (Pentium 4 SSE, QCDOC)
Staggered Dslash Op (QCDOC)
Level 3
Optimized Dirac Operators
Sparse Matrix Inverters, etc
Level 2
QDP (QCD Data Parallel)
Lattice Wide Operations
Data Shifts
QIO
XML
LIME
C/C++ implementations (QDP/QDP++)
Parallel using QMP
First Release:
Record Based Binary Files
(similar to DIME encapsulation)
Level 1
QLA ( QCD Linear Algebra – site ops)
QMP ( QCD Message Passing)
implemented over MPI, MVIA Gig–E, GM, QCDOC–SCU

Fig. 1: The SciDAC Software Hierarchy

### Level 3 and BAGEL

Level 3 of the SciDAC hierarchy is not yet as mature as the other levels. Hence Chroma uses optimised Dslash operators from a variety of sources. The Pentium 4 SSE assembler versions used are the ones developed at the JLAB [5].

For RISC architectures such as the QCDOC and POWER based systems we have interfaced to code produced by the BAGEL assembly generator developed by Peter Boyle [6].

The BAGEL assembly generator produces highly optimised assembler kernels for RISC microprocessors by modelling the CPU pipelines and taking into account the latencies and execution times of the CPU functional units. By considering the bandwidth between the CPU, cache and memory, BAGEL can also encode aggressive prefetching schemes resulting in extremely fast code.

While BAGEL is not SciDAC software, it is in fact a very versatile tool that can be used to generate operators for SciDAC's level 3. It can also be used to generate optimised code at other levels of the hierarchy. The source code for BAGEL will be available for download in the future.

## QDP++

QDP++ is a C++ implementation of the QDP level of the SciDAC hierarchy, which Chroma uses as its foundation. QDP++ defines lattice–wide types and allows lattice–wide expressions. It has also been integrated with the QIO framework and with the XPathReader and XMLWriter modules to provide XML based I/O (e.g. for reading of parameters, or creating QCDML markup [7] in the future).

### Lattice–Wide Types

QDP++ models the tensor product structure of LQCD objects through a series of nested templates. E.g: the indices of lattice fermion fields can follow the structure:

$$\text{sites} \otimes \text{spins} \otimes \text{colours} \otimes \text{complex} \otimes \text{type of complex components}$$

QDP++ would model this through the following C++ templated type:

```
OLattice< PSpinVector< PColorVector< RComplex < PScalar < REAL> >, Nc >, Ns > >
```

where `PScalar<REAL>` is the type of the complex components. The constant `REAL` is defined as either `float` or `double` while `Nc` and `Ns` are the numbers of spin and colour components. These constants are defined during configuration.

QDP++ operates on such types by recursing down the templates. To multiply the above type by a scalar, one would first loop through the indices of the `OLattice` type and for each one call the multiply operation for the `PSpinVector` type and so on.

The order of the templates is not fixed in principle. One could equally well exchange the ordering of the indices in order to take advantage of particular parallel architectures.

In order to save fingers and brains, the above type is usually aliased to a shorthand C++ type using a `typedef` statement. The type above is usually referred to as a `LatticeFermion`.

### Lattice–Wide Operations, Expressions, PETE

By overriding operators in C++, QDP++ provides lattice–wide arithmetic. For example the AXPY operation

$$z \leftarrow \alpha x + y$$

where $z, x$ and $y$ are lattice fermion fields and $\alpha$ is a scalar, would be coded as:

```
z = a*x + y
```

where `z`, `x` and `y` are of type `LatticeFermion` and `a` is e.g.: a `Real`.

In order to eliminate temporaries in such expressions, QDP++ uses Portable Expression Template Extensions (PETE). The C++ binary operators are redefined to transform the expression into an **expression template type**. On assignment, the expression template and the left hand side are given as arguments to an *evaluate()* function. This function now knows it has a three–operand operation to perform, and has references to all 3 operands and the result. **All this is done at *compile time* through the template instantiation mechanism**. We illustrate the main points of the idea in Fig. 2.

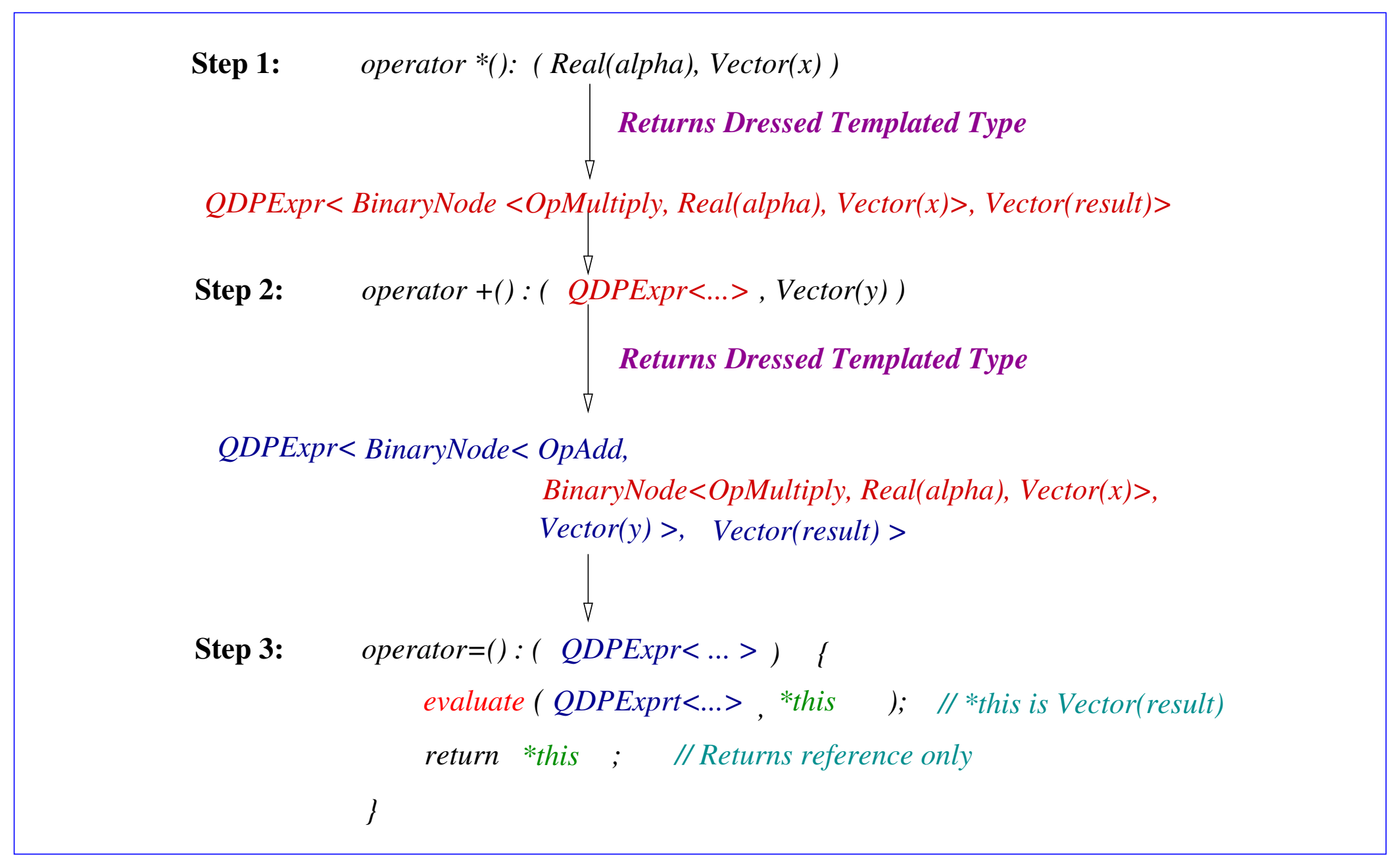
Step 1: operator *(): ( Real(alpha), Vector(x) )
Returns Dressed Templated Type
QDPExpr< BinaryNode <OpMultiply, Real(alpha), Vector(x)>, Vector(result)>
Step 2: operator +() : ( QDPExpr<...> , Vector(y) )
Returns Dressed Templated Type
QDPExpr< BinaryNode< OpAdd,
BinaryNode<OpMultiply, Real(alpha), Vector(x)>,
Vector(y) >, Vector(result) >
Step 3: operator=() : ( QDPExpr< ... > ) {
evaluate ( QDPExprt<...> , *this ); // *this is Vector(result)
return *this ; // Returns reference only
}

Fig. 2: Expression transformation a la PETE

### QDP++ Optimisation

We can further **specialise** the *evaluate()* functions for specific expression templates, and make these call optimised subroutines, e.g. fast AXPY like functions. This approach is in some sense of limited use, since only a few hand–picked expressions are optimised. However, in practice, only a small number of routines (e.g. Level 1 BLAS) are of greatest utility and of most common use. To this end we have optimised real and complex AXPY routines, inner products and vector norm operations.

It may be possible to optimise more general expressions by applying PETE techniques at all levels of our template hierarchy rather than just at the `OLattice` level as we do now. This is current work in progress.

## Chroma

### Code Base Content Overview

Chroma developed to service the current needs of the LHPC and UKQCD collaborations, which have mostly included *spectroscopy, decay constant, nucleon form factor* and *structure function moment* calculations. Another branch of development was related to the investigation of *chiral fermion actions*. Hence the current version of the code contains *Wilson, Domain Wall* and *Overlap fermion operators* and *numerous inverters* to compute *quark propagators* and *sequential propagators*. Code also exists to compute *hadronic 2-point* and *3-point* correlation functions. A number of UKQCD researchers have also written an *ASqTAD fermion operator*, in the staggered library as well as code to compute *disconnected diagrams*.

### Factory Functions, Linear Operators, Generic Solvers

In order to support generic iterative solver algorithms Chroma defines an abstract linear operator (`LinearOperator`) base class. Actual linear operators are derived from this class and implement its virtual function interface. There are also fermion action (`FermAct`) classes which provide *factory functions* to produce appropriate Dirac operators (see Fig. 3). These Dirac operators use *optimised Dslash routines* under the hood where possible. The linear operators can be given to templated inverters as input. These inverters are generic in the sense that they can be used for a variety of fermion actions. The same Conjugate Gradient (CG) code can be used for the Wilson action as for the Staggered and the Overlap actions. We show the code for a CG Solver in Fig 4.

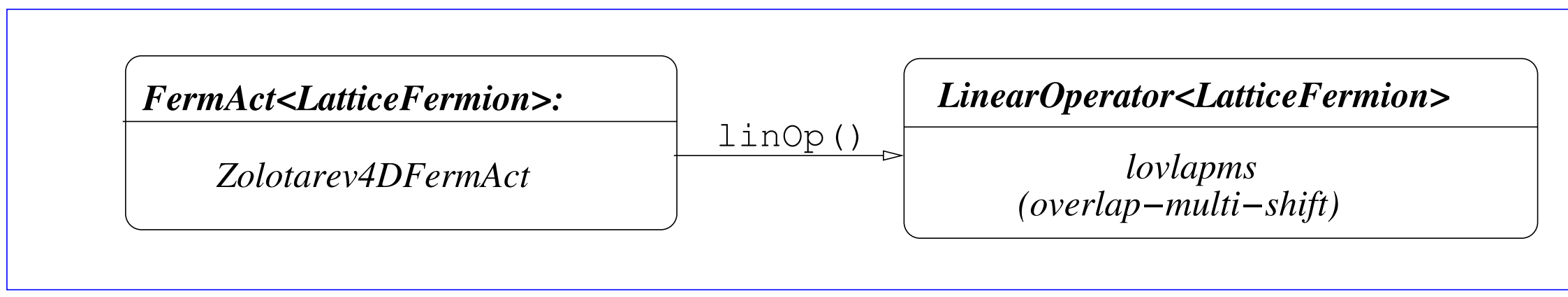

Fig. 3: Factory Functions produce Linear Operators for Overlap

### Virtual Functions, Category Defaults

The *linOp()* factory function is an example of a *virtual function* defined in the base *FermAct* class. *FermAct* defines other virtual functions which derived classes may need to implement (if they are *pure virtual*) or override. Examples of these are *qprop()* which computes the quark propagator for a given action and *dsdu()* whose job is to compute the fermionic force term for the action in a Molecular Dynamics evolution algorithm.

Virtual functions allow us to maintain the same *look and feel* across our fermion action classes, to write generic algorithms and to provide default implementations in the base classes (In the Aldor [8] language these would be called *category default* functions)

```
template<typename T>
void InvCG1_a(const LinearOperator<T>& A, const T& chi, T& psi,
      const Real& RsdCG, int MaxCG, int& n_count)
{
  const OrderedSubset& s = A.subset();  // Which part of vectors to operate on
  Real rsd_sq = (RsdCG * RsdCG) * Real(norm2(chi,s)); // Target rel. residue

  T tmp1;   T r;   // Temporary and residual vector

  A(tmp1, psi, PLUS);                // Work out chi - A psi_0
  r[s] = chi - tmp1;                 // Work only on subset

  T p;
  p[s] = r;                         // Conjugate search vector
  Double cp = norm2(r, s);          // Initial Residue

  if ( toBool(cp  <=  rsd_sq) ) {   // Catch RHS=0 case
    n_count = 0; psi=zero;
    return;
  }

  T ap; Real b; Double c; Complex a; Real d;

  for(int k = 1; k <= MaxCG; ++k) { // CG Iterations start here
    c = p;
    A(ap, p, PLUS);
    d = innerProductReal(p, ap, s);           // < p , Ap >
    a = Real(c)/d;
    psi[s] += a * p;                          // update psi
    r[s] -= a * ap;                           // update r
    cp = norm2(r, s);

    if ( toBool(cp  <=  rsd_sq) ) {          // Check convergence
      n_count = k; return;                   // and return if done
    }

    b = Real(cp) / Real(c);
    p[s] = r + b*p;                          // update p
  }
  n_count = MaxCG;
  QDP_error_exit("too many CG iterations: count = %d", n_count);
}
```

Fig. 4: Conjugate Gradients for Hermitian Matrix A in Chroma

### QDP++/Chroma Efficiency

By optimising QDP++ as discussed previously and using high–performance Wilson Dslash kernels in our linear operators we have managed to achieve high performance on workstations, Pentium 4 clusters and the QCDOC.

Fig. 5 shows some performance measurements with an even-odd preconditioned Wilson Dirac operator on 4 nodes of the QCDOC. In this test we have used the assembler Dslash from BAGEL. We show the performance of the Dslash operator, the even–odd Wilson Dirac operator, the CG algorithm from Fig 4, and the performance of a *super solver*. The super solver is a CG algorithm, where we have replaced our *LinearOperator* with direct calls to the *Dslash* routine, and have fused the application of the Dirac Matrix with the computation of the norm of the result where possible. The tests were carried out in single precision.

The Dslash routine (from BAGEL) was designed to amortise communications latency at a local lattice size of 256 sites ($V = 4^4$). This can be seen in Fig. 5. The cost of the additional vector operations needed to combine the Dslash applications into the even-odd Wilson operator appear as the difference between the red and black bars.

We note that the difference between the CG algorithm of Fig 4. and the "super solver" is about 2-3% of peak over the whole range of volumes. This shows that the overhead from QDP++ expressions is very small.

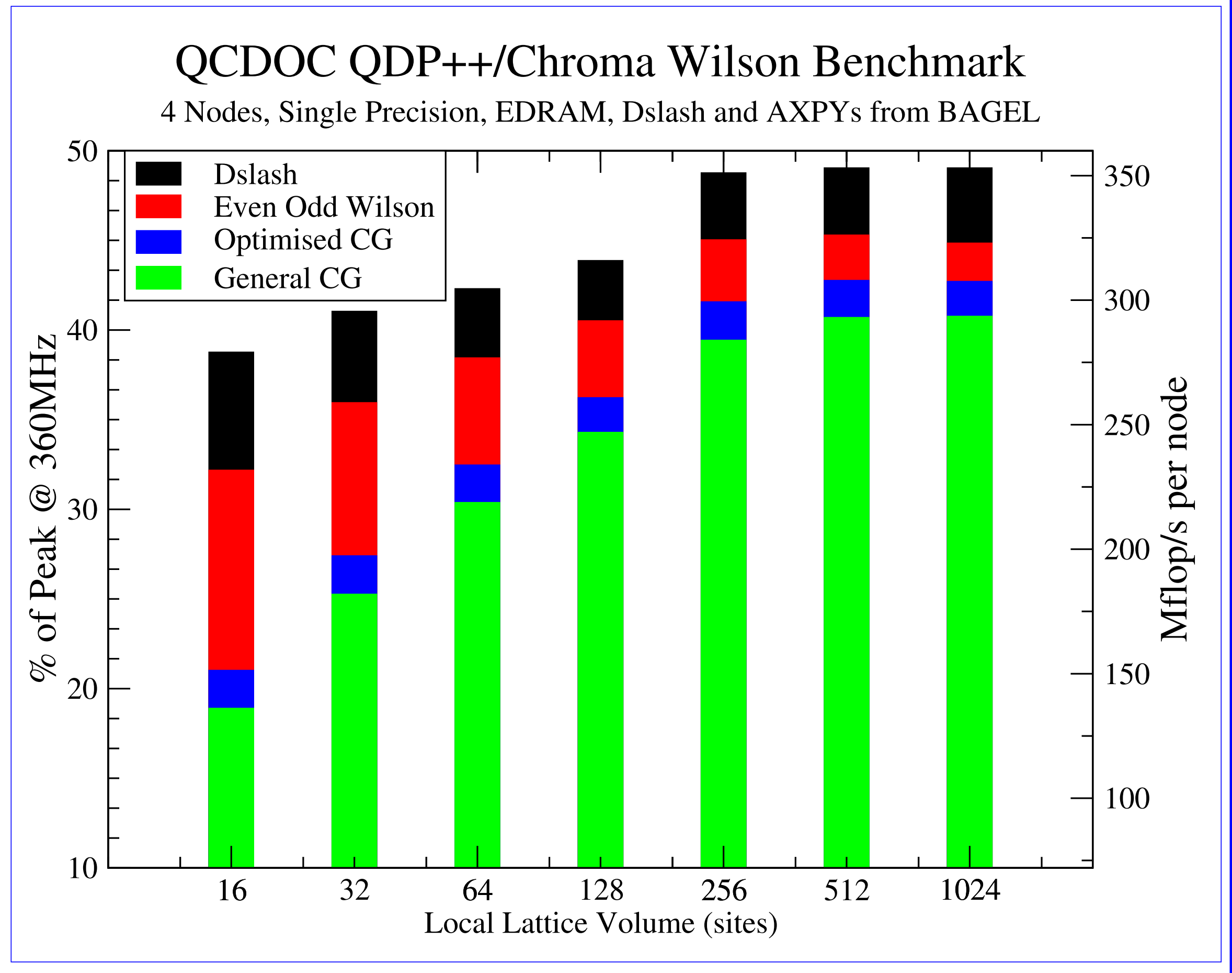

Fig. 5: Single Precision Performance on the QCDOC (using BAGEL Dslash and AXPYs)

## Overlap Fermions in Chroma

QDP++ expressions and the structure of Chroma have made it ideal to try out algorithmic ideas. We mention briefly some work we carried out to investigate the Overlap operator:

We have implementations of the Overlap [9] operator with the sign function approximated as a partial fraction expansion of a rational polynomial whose coefficients are given by Zolotarev's theorem [10]. We calculate the coefficients on the fly [11]. Both single pass and double pass [12] implementations of the 4D operator exist. We also have a 5D operator which expresses the Overlap operator as a continued fraction (5DCF) [13].

On the inversion front we have tried most of the suggestions of the Wuppertal group [14]. We have implemented the SUMR [14,15] solver algorithm – for single and multiple masses (a trivial extension) – as well as relaxed versions of both SUMR and CG. Our fastest current algorithm for the 4D case, however, is the RelGMRESR algorithm discussed in the second paper of [14].

Our 5D algorithmic studies are still in infancy. Initial results suggest that inverting the 5DCF operator needs fewer applications of $\gamma_5 D_W$ than the 4D nested CG inversion. However, when projecting eigenvalues of $\gamma_5 D_W$ the 5D method calls for more projections thus clouding the advantage. Currently we are looking for a good preconditioner for the 5DCF operator and investigating other 5D formulations. Even-odd preconditioning may be possible, and using the GMRESR algorithm of [14,16] may allow for other preconditioners based on approximate inverses of the operator.

## Conclusions and Future Work

We have discussed the SciDAC software hierarchy and elements of the Chroma software suite for lattice QCD. We have shown how QDP++ and Chroma may be optimised in order to achieve high efficiency through template expressions and by interfacing cleanly with third–party high–performance libraries such as the assembly code produced by BAGEL. We have also illustrated the ease of implementing new solver algorithms.

Due to lack of space we have had to gloss over several other nice features of Chroma and QDP++: the QIO framework and the XML I/O system were just briefly mentioned, as were the many applications already existing in the code–base. We have not even touched upon our build system which uses the GNU Autotools allowing for easy configuration and building.

We believe Chroma has a bright future. Already the LHPC and UKQCD collaborations are using it for production calculations on a variety of clusters and it has been ported and partially optimised for the QCDOC.

Our future plans include the implementation of various Hybrid Monte Carlo algorithms (HMC, RHMC) and the provision of a full Dynamical Fermion code suite.

We should finish off by saying that QMP, QDP++ and Chroma are *freely available through anonymous CVS*. The CVSROOT is:

```
:pserver:anonymous@cvs.jlab.org:/group/lattice/cvsroot
```

See [1] for details. QMP, QDP++ and Chroma are open source. Feel free to contribute...

## References

[1] http://www.jlab.org/~edwards/chroma
[2] http://www.jlab.org/~dgr/lhpc
[3] http://www.lqcd.org/scidac
[4] http://www.ph.ed.ac.uk/ukqcd
[5] C. McClendon, JLAB–THY–01–29, http://www.jlab.org/~edwards/qcdapi/reports/dslash_p4.pdf
[6] Contact Peter Boyle: pab@phys.columbia.edu, paboyle@ph.ed.ac.uk
[7] http://www.lqcd.org/ildg/tiki–index.php?page=MetaData
[8] http://www.aldor.org
[9] R. Narayanan, H. Neuberger, Phys. Lett. B302, 62, (1993), hep–lat/9212019
[10] P. P. Petrushev, V. A. Popov, "Rational Approximation of Real Functions". Cambridge Univ. Press, Cambridge 1987
[11] A. D. Kennedy, hep–lat/0402038
[12] H. Neuberger, Int. J. Mod. Phys C10 (1999), 1051–1058, hep–lat/9811019
T–W Chiu, T–H Hsieh, Phys. Rev. E68 (2003) 066704, hep–lat/0306025
[13] A. Borici, et al. Nucl. Phys. Proc. Suppl. 106(2002) 757–759, hep–lat/0110070, U. Wenger, hep–lat/0403003
[14] G. Arnold. et. al. hep–lat/0311025, N. Cundy et. al. hep–lat/0405003
[15] C. F. Jagels, L. Reichel, Numerical Linear Algebra with Applications, Vol1(6),555–570 (1994)
[16] H. van der Vorst, C. Vuik, Numerical Linear Algebra with Applications, Vol1(4), 369–386, 1994